Say $p_1, \dots, p_n$ prob of keys $K_1 < \cdots < K_n$ and $q_0, q_1, \dots, q_n$ prob of missing keys between

So $\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$

problem is to find a binary tree that minimizes $\sum_{i=1}^{n} p_i (\text{level}(i)+1) + \sum_{i=0}^{n} q_i (\text{level}(i)+1) = 1 + \sum_{i=1}^{n} p_i \,\text{level}(i) + \sum_{i=0}^{n} q_i \,\text{level}(i) =: \text{cost}$ (A)

level(8) = 2 in

access data @ key i, level(i) so +1

optimal = minimize cost

optimal substructure: all proper subtrees of optimal tree are optimal → key obs. is that any improvement to a subtree leads to an improvement in the whole tree (& conversely)

So should try to find larger & larger optimal subtrees

Subproblem: find optimal subtree containing keys $K_i, \dots, K_j$ w/ prob $(p_i, \dots, p_j; q_{i-1}, \dots, q_j)$ (key probs; untranslatable words / dummy keys)

let $c(i,j)$ = cost of searching this optimal tree → note: $c(i, i-1) = q_{i-1}$ since only a single missing key

When $j \geq i$, need to select a root K_r from $K_i, \dots, K_j$ & make K_r optimal

note depth of these nodes increased by 1

thus, cost of subtree w/ keys $K_i, \dots, K_j$ increases by $w(i,j) := \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$ by using (A)

Hence can write $c(i,j) = p_r + c(i, r-1) + w(i, r-1) + c(r+1, j) + w(r+1, j)$

(chose K_r as root; cost of left; cost of right; subtree by itself + incr. amount due to depth +=1)

$= (w(i, r-1) + p_r + w(r+1, j)) + c(i, r-1) + c(r+1, j)$

$= w(i,j) + c(i, r-1) + c(r+1, j)$

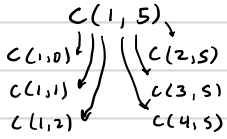
So now we just need to pick the optimal root when we don't already know it:

$$c(i,j) = \begin{cases} q_{j-1} & \text{if } j = i-1 \\ \min_{i \leq r \leq j} \left(c(i, r-1) + c(r+1, j) + w(i,j) \right) & j \geq i \end{cases}$$

observe: 1. Keep track of roots picked to get full tree

2. This is a recursive def'n of $c(i,j)$ that uses previous subproblems ∴ DP

Subproblem graph:



note: subproblems are contiguous index ranges: $c(i,j) \rightarrow 1 \leq i \leq j \leq n$

Turn it into a DP: Store the $c(\cdot,\cdot)$ values in an array: $c[1 \dots n+1; 0 \dots n]$ (take care of missing keys) ← will only populate/use entries where $j \geq i-1$

Keep track of root nodes in an array: $root[1 \dots n; 1 \dots n]$ ← will only use entries $1 \leq i \leq j \leq n$

Don't recompute w each time, store them in an array $w[1 \dots n+1; 0 \dots n]$ ← note: base case $w[i, i-1] = q_{i-1}$ $\forall\ 1 \leq i \leq n+1$

for $j \geq i$, $w[i,j] = w[i, j-1] + p_j + q_j$ (from (A))

So each value of w computed in $\Theta(1)$ time

$P = (p_1, \ldots, p_n)$, $q = (q_0, \ldots, q_n)$

```
def Optimal_BST(p, q, n)
initialize empty arrays c[1..n+1, 0..n], w[1..n+1, 0..n], root(1..n, 1..n)
for i = 1:n+1
    c[i, i-1] = q_{i-1},  w[i, i-1] = q_{i-1}        <- basic cases for trees w/ only missing keys
for L = 1:n
    for i = 1 : n-L+1
        j = i + L - 1                                 <- i.e. j ≥ i
        c[i, j] = ∞
        w[i, j] = w[i, j-1] + p_j + q_j               <- use recurrence w/ precomputed values
        for r = i:j                                   <- try all root nodes & pick the least cost one (& store it)
            t = c[i, r-1] + c[r+1, j] + w[i, j]
            if t < c[i, j]
                c[i, j] = t,  root[i, j] = r
return e and root
```

$\Theta(n^3)$ time (but can cut it to $O(n^2)$ by proving a property re: which roots to check & modifying the alg.)

eg

i	0	1	2	3	4	5
p_i		.15	.10	.05	.10	.20
q_i	.05	.10	.05	.05	.05	.10

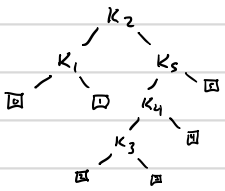
initialize

first step through outer loop: $L = 1$, $i = 1$, $j = 1$

$r = 1 \Rightarrow t = c[1,0] + c[2,1] + w[1,1]$

$= .05 + .10 + .30 = .45$

$root[1,1] = 1$



cost = 2.75

c

i \ j	0	1	2	3	4	5
1		.05	.45			2.75
2			.10			
3				.05		
4					.05	
5						.10
6						

w

i \ j	0	1	2	3	4	5
1		.05	.30			1
2			.10			
3				.05		
4					.05	
5						.10
6						

won't use these

root

i \ j	1	2	3	4	5
1	1	1	2	2	2
2		2	2	2	4
3			3	4	5
4				4	5
5					5

root[i, j] = root of subtree w/ keys i...j

Final example: All pairs shortest paths

Say we have a weighted, directed graph $G=(V,E)$ w/ no negative weight cycles.

How to compute shortest path (u,v) $\forall u,v \in V$?

Bellman-Ford: $O(|V| \cdot |E|)$ to compute shortest path $(u, \cdot)$ for fixed $u \in V$

So could run Bellman-Ford on every vertex: $O(|V|^2 |E|)$ → could be inefficient if $|E|$ large (eg $|E| \approx |V|^2$)

Let's try a DP approach: want optimal, overlapping substruct → what subproblems to choose?

↑ Clear since shortest$(u,v) = u \to w_1 \to \cdots \to w_n \to v$

$\Rightarrow$ shortest$(w_1, v) = w_1 \to \cdots w_n \to v$

Observe: Say $V = \{1, \ldots, n\}$, $u, v \in V$ consider subproblem of finding shortest path $u \rightsquigarrow v$ using only intermediate vertices from $\{1, \ldots, k\} \subseteq V$, $k \leq n$. want to relate to same problem but w/ $\{1, \ldots, k-1\}$

Using nodes $\{1, \ldots, k\}$, say shortest_path$(u,v,k) = p$ ← use only vertices $\{1, \ldots, k\}$

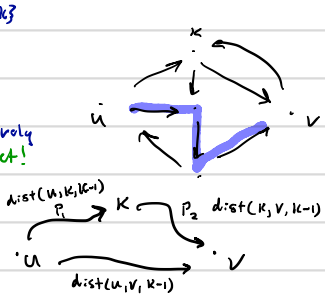
If $k \notin p$, then shortest_path(u,v,k) = shortest_path$(u,v,k-1)$

If $k \in p$, consider decomposition $p = u \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} v$ ← p_1, p_2 optimal for (u,k), (k,v) respectively optimal substruct!

$k \notin p_1, p_2$ as p is a simple path, hence

$p_1 = u \rightsquigarrow k$ and $p_2 = k \rightsquigarrow v$ only use vertices from $\{1, \ldots, k-1\}$!

So then only need check if p is better or worse than shortest$(u,v,k-1)$:



Using k is better if: (dist. of path $p = u \rightsquigarrow k \rightsquigarrow v$ using $\{1, \ldots, k\}$)

$$dist(u,k,k-1) + dist(k,v,k-1) < dist(u,v,k-1)$$

hence: $$dist(u,v,k) = \min\{ dist(u,k,k-1) + dist(k,v,k-1),\ dist(u,v,k-1) \}$$

↑ depends on 3 subproblems

Note: For $w_{u,v}$ the edge weights, $d(u,v,0) = w_{u,v}$ ← use no other intermed. vertices

Putting this together, we get

← $G=(V,E)$, $V=\{1,\ldots,n\}$, $W = \{w_{u,v} \ \forall u,v \in V\}$

```
def Floyd_Warshall(G, W)
    for u=1,...,n
        for v=1,...,n                          } O(|V|^2)
            dist(u,v,0) = w_uv
    for k=1,...,n
        for u=1,...,n
            for v=1,...,n                      } O(|V|^3)
                dist(u,v,k) = min{ dist(u,k,k-1) + dist(k,v,k-1), dist(u,v,k-1) }
```

$\therefore O(|V|^3)$ (better than $O(|V|^2|E|)$ since $|E| \leq |V|^2$)

Linear Programming

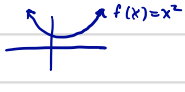
eg limited supplies, time, etc

$c_1(x_1,\dots,x_n) \le b_1$
$c_2(x_1,\dots,x_n) \le b_2$
$\vdots$
$c_m(x_1,\dots,x_n) \le b_m$

Optimization: Say you want to maximize some function $f(\vec{x})$ subject to some constraints $C(\vec{x}) \le \vec{b}$

(profit, production, cost, time; $\vec{x} = (x_1,\dots,x_n)$)

If you know very little about the function $f(\vec{x})$ (eg stock prices) then this is a difficult problem called general purpose global optimization. Approaches exist (eg Nelder-Meade, Differential Evolution, Simulated annealing) but performance can vary wildly.



If you know a bit more about f & C, say that they are convex ($f(a\cdot x + b\cdot y) \le a f(x) + b f(y)$, $a,b \in [0,1]$) then the problem is easier (but it's a very large & still active field) called convex optimization. Approaches include: descent methods, Newton's method, barrier method, prime-dual interior point, etc).

Stuff in between includes: quadratic programming (QP), 2nd order cone prog (SOCP), semi-definite prog. (SDP), etc, etc.

If, however, you assume the nicest possible non-trivial case: f and C are linear functions (eg $f(\vec{x}) = a_1x_1 + \dots + a_nx_n$) then you end up w/ Linear programming. We consider that case here.

eg 168 hours in a week. Allocate to studying (S), fun/parties (P), everything else like sleep, eating, hygene, etc (E).

To survive, need @ least $E \ge 56$ (8 hrs/day)

To stay sane, need $P+E \ge 70$ (70 hrs/week on fun & everything else)

To pass courses, need $S \ge 60$ (60 hrs/week in classes & studying), but more if too much time spent on fun or not enough sleep:

$2S + E - 3P \ge 150$

↖ decrease E or increase P & you need to increase S

Say our happiness can be expressed as: $2P+E$ ← this is our objective function $f(S,P,E) = 2P+E$

Want to maximize happiness:

$$\max_{S,P,E} \; 2P+E \quad \leftarrow \text{obj. func.}$$

Subject to

$$E \ge 56$$
$$S \ge 60$$
$$2S+E-3P \ge 150$$
$$P+E \ge 70$$
$$S,P,E \ge 0 \quad \text{(can't spend negative time on stuff)}$$
$$S,P,E \le 168 \quad \text{(total hours in a week)}$$

} constraints

LHS all linear funcs

RHS all constants

In order to solve this, need to understand what's going on.

Let's consider a simpler example where we can visualize things.

eg Say $\max_{x,y}$ $x+2y$

$x \le 20$

$y \le 30$

$x+y \le 40$

$x, y \ge 0$

eg 2 products to sell, x and y

y is worth 2 times x → profit = $x+2y$

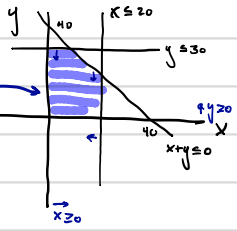
demand for $x \le 20$ units

" " $y \le 30$ units

work force can only make 40 units daily → $x+y \le 40$

can't have negative units

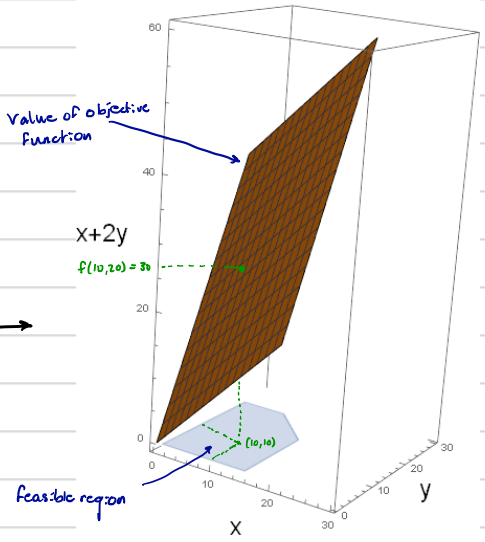
Since 2 vars, let's plot it:



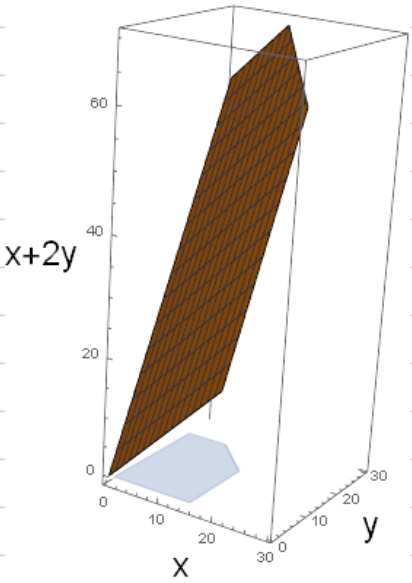
Constraints tell us to pick a soln (x,y) in here

Call the set of all points (x,y) satisfying the constraints the feasible region (could be empty)

The objective function can be plotted in 3D: $z = f(x,y) = x+2y$. Visualizing this →



← only plot f(x,y) above feasible region



Observe: Since objective function is linear (∃ feasible region is convex) in some direction the objective function will be monotonically/strictly maximally (↑) increasing. Going this direction, you'll eventually hit the boundary of the feasible region. ∴ only the boundary matters.

Once at the boundary, moving left or right will either:

a) decrease the objective func → don't go this way

b) keep the obj. func const → follow it to a vertex

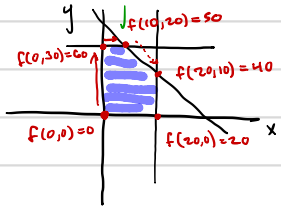
c) increase the obj. func. → " " " " "

Once @ the vertex, this will be the maximum (moving any further will take you in a dir opposite to (↑))

Theorem: For a linear program s.t. the feasible region is non-empty and bounded, the maximum will be attained @ some vertex of the feasible region.

Algorithm idea: Hill climbing (i.e. simplex method of Dantzig 1947)

Start @ vertex, look @ adjacent vertices, move in dir. of largest increase to obj. func.



another eg Typically the hardest part is formulating the right LP

Operations research

4 factories making cars, per car, statistics are

labor agreement says need at least 400 cars @ plant 3.

We have 3300 hours of labor avail

4000 units of material

	factory	labor	materials	pollution
	1	2	3	15
	2	3	4	10
	3	4	5	9
	4	5	6	7

can produce @ most 12,000 units of pollution. How to max. the # of cars made?

1) What are the vars?

$x_1, \dots, x_4$ so x_i = # cars @ factory i

2) What's the objective?

$x_1+x_2+x_3+x_4$ = total # of cars produced

3) What are the constraints?

$x_1 \geq 0, \dots, x_4 \geq 0$

$x_3 \geq 400$

$2x_1+3x_2+4x_3+5x_4 \leq 3300$ labor hours

$3x_1+4x_2+5x_3+6x_4 \leq 4000$ materials

$15x_1+10x_2+9x_3+7x_4 \leq 12000$ pollution

$\therefore$ problem is $\max \sum_{i=1}^{4} x_i$ s.t. ↙ those constraints

Standard Form

Def'n an LP in standard form is one written as:

For $n \in \mathbb{N}$, $c_1, \dots, c_n \in \mathbb{R}$, for $m \in \mathbb{N}$, $b_1, \dots, b_m \in \mathbb{R}$, and for $i = 1, \dots, m$, $j = 1, \dots, n$, $a_{ij} \in \mathbb{R}$, with variables $x_1, \dots, x_n$, the LP is

$$\text{maximize} \sum_{j=1}^{n} c_j x_j$$

$$\text{Subject to} \sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \text{for } i = 1, \dots, m$$

$$x_j \geq 0 \quad \text{for } j = 1, \dots, n$$

$\Leftrightarrow$ equivalently

$x = (x_1, \dots, x_n)$

For $c = (c_1, \dots, c_n)$ and $b = (b_1, \dots, b_m)$ vectors, $A = (a_{ij})$ a matrix

$$\text{maximize } c^T x$$

$$\text{Subject to } Ax \leq b$$

$$x \geq 0$$

↖ Standard Form

Note: Not all LP's come in standard form, but all can be reduced to it by doing the following:

1) minimize to maximize

$$\min \sum_{j=1}^{n} c_j x_j \Leftrightarrow \max -\sum_{j=1}^{n} c_j x_j$$

negate obj. func $\rightarrow$ use max

2) equality constraints

$$\max c^T x \Leftrightarrow \max c^T x$$

s.t. $x_1+x_2 = 7$ ↙ since this is true iff

s.t. $x_1+x_2 \leq 7$ $x_1+x_2 = 7$

$x_1+x_2 \geq 7$

Split into 2 inequalities

3) inequality sign the wrong way around

$\max\ x_1+x_2+x_3$ s.t. $2x_1+x_2 \geq 0$ $\Leftrightarrow$ $\max\ x_1+x_2+x_3$ s.t. $-2x_1-x_2 \leq 0$ — mult. constraint w/ -1

4) missing non-neg. constraint

Say LP has var. x unconstrained. Can write $x = x^+ - x^-$ where $x^+ \geq 0$ and $x^- \geq 0$ — split one var. into 2

eg LP

$\min\ -2x_1+3x_2$ s.t. $x_1+x_2=7$, $x_1-2x_2 \leq 4$, $x_1 \geq 0$

$\to \max\ 2x_1+3x_2$ s.t. $x_1+x_2=7$, $x_1-2x_2 \leq 4$, $x_1 \geq 0$

$\to \max\ 2x_1+3x_2$ s.t. $x_1+x_2 \leq 7$, $x_1+x_2 \geq 7$, $x_1-2x_2 \leq 4$, $x_1 \geq 0$

$\to \max\ 2x_1+3(x_2^+-x_2^-)$ s.t. $x_1+x_2^+-x_2^- \leq 7$, $x_1+x_2^+-x_2^- \geq 7$, $x_1-2x_2^++2x_2^- \leq 4$, $x_1 \geq 0$, $x_2^+ \geq 0$, $x_2^- \geq 0$

5) Scaling Replace $x \geq c$ w/ $(x-c) \geq 0$

↖ new var $x' := x-c$ $\Rightarrow x'+c = x$

$\max\ 2x+y$ s.t. $x+y \leq 30$, $y \geq 0$, $x \geq 10$ $\to$ $\max\ 2(x'+10)+y$ s.t. $x'+10+y \leq 30 \Rightarrow x'+y \leq 20$, $x', y \geq 0$

$\max\ (2,3,-3)^T \begin{pmatrix} x_1 \\ x_2^+ \\ x_2^- \end{pmatrix}$ s.t. $\begin{pmatrix} 1 & 1 & -1 \\ -1 & -1 & 1 \\ 1 & -2 & 2 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2^+ \\ x_2^- \end{pmatrix} \leq \begin{pmatrix} 7 \\ -7 \\ 4 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ (A, x, b)

$\max\ 2x_1+3(x_2^+-x_2^-)$ s.t. $x_1+x_2^+-x_2^- \leq 7$, $-x_1-x_2^++x_2^- \leq -7$, $x_1-2x_2^++2x_2^- \leq 4$, $-x_1 \leq 0$, $-x_2^+ \leq 0$, $-x_2^- \leq 0$

Sometimes convenient to go the other way:

$\sum_{i=1}^{n} a_i x_i \leq b \Leftrightarrow \sum_{i=1}^{n} a_i x_i + s = b$, $s \geq 0$

this means b is bigger by some positive value. Call that value s $\Rightarrow$

maximize $c^T x$
subject to $Ax = b$
$x \geq 0$

← alt. Standard form

Application Shortest path

$G=(V,E)$, $w: E \to \mathbb{R}$, want shortest(s,t) for given $s,t \in V$, but as an LP

Need: variables & inequalities. Recall Bellman Ford: @ the end, calculates d_v $\forall v \in V$ s.t. $\forall (u,v) \in E$, $d_v \leq d_u + w(u,v)$

Thus, we have:

$$d_v = \min_{\substack{u \in V \\ (u,v) \in E}} \{d_u + w(u,v)\}$$ ← i.e. d_v is the largest value s.t. $d_v \leq d_u + w(u,v)$ $\forall u \in V$

This suggests:

maximize d_t
subject to $d_v \leq d_u + w(u,v)$ $\forall (u,v) \in E$
$d_s = 0$

$|V|$ variables
$|E|+1$ constraints

Hence we have a reduction of shortest path ⇝ LP problem

Let's see another example...

Application: *Network flow*

$G=(V,E)$, $s,t \in V$ source & sink, $\forall e \in E$, $c_e > 0$ capacities

Recall: flow $f=(f_e)_{e \in E}$ s.t.

1. $0 \le f_e \le c_e \quad \forall e \in E$ (don't exceed capacity)
2. $\sum_{(w,v) \in E} f_{wv} = \sum_{(v,u) \in E} f_{vu}$ (flow in = flow out, conservation)

↝

$$\max \sum_{(s,u) \in E} f_{su}$$
$$\text{s.t.} \quad f_e \le c_e \quad \forall e \in E$$
$$f_e \ge 0 \quad \forall e \in E$$
$$\sum_{(w,v) \in E} f_{wv} - \sum_{(v,u) \in E} f_{vu} = 0 \quad \forall v \in V$$

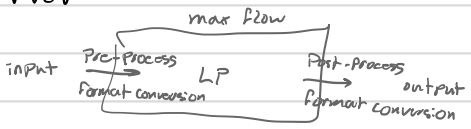
↓ Variables are the flow values

and max (s,t)-cut problem

It's an LP!

∴ max-flow problem reduces to an LP problem

Volume of flow $V(f) = \sum_{(s,u) \in E} f_{su}$



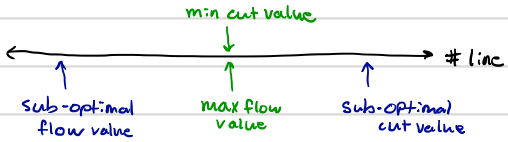
Even more remarkable, if you follow the simplex algorithm for this LP, you get exactly the Ford-Fulkerson algorithm!

Note: These sorts of reductions are key to algorithm theory. The text has more examples such as

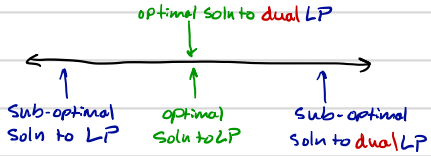
Bipartite matching $\xrightarrow{\text{reduces to}}$ max flow $\xrightarrow{\text{reduces to}}$ LP

Duality Similar to what we saw in Ford-Fulkerson re: correspondence between flows & cuts (in particular $V(f) \le C(A,B)$ for any s-t cut (A,B), and f^{max} optimal $\Rightarrow \exists$ s-t cut (A^*,B^*) s.t. $V(f^{max}) = C(A^*,B^*)$), there is a similar duality for LP's.

Recall the basic idea:



We shall see that a similar thing happens w/ LP's!



Motivation: eg LP

$$\max \; X_1 + 2X_2$$
$$\text{s.t.} \quad X_1 \le 20 \quad \leftarrow \text{1 of these}$$
$$X_2 \le 30 \quad \leftarrow \text{2 of these}$$
$$X_1 + X_2 \le 40 \quad \leftarrow \text{one of these}$$
$$X_1, X_2 \ge 0$$

Note! add mult. of some inequalities: $X_1 + 2X_2 \le 80$ hence, we know that the soln to the LP can be at most 80. But can we make this upper bound be tighter?

Yes! Try adding other ineq: $X_1 + 2X_2 \le 70$ so now we know the soln can be @ most 70.

What's the actual soln? $X_1 = 10$, $X_2 = 30 \Rightarrow X_1 + 2X_2 = 70$. So we've achieved our upper bound from ! !

This isn't a fluke, but rather an instance of a general fact called **duality**.

The basic idea is: find the best/tightest upper bound on the soln to the LP by adding (pos.) multiples of the inequalities together (so obj. func. appears on LHS of $\le$) then soln to LP will achieve this best upper bound.

But, how do you find this best/tightest upper bound?